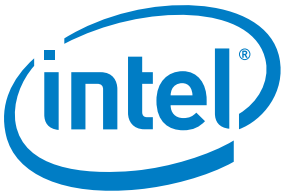


A Unified Interface for Benchmark Tools on the Intel® Xeon Phi™ Processor X200 Product Family

and

The Intel® Xeon Phi™ Coprocessor X200 Product Family

Micperf User Guide

March 2017

Copyright © 2017 Intel Corporation

All Rights Reserved

Revision: 1.15

World Wide Web: http://www.intel.com

Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

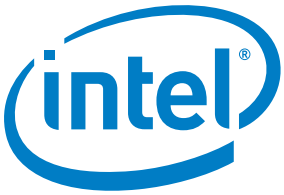
All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

*Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.



Contents

1 Introduction 6
- 1.1 Organization 7
- 1.2 Conventions and Symbols 7

2 Novice User: Benchmark Execution 8
- 2.1 Benchmark Selection 9
- 2.2 Memory Selection (Only available for the Intel® Xeon Phi™ Processor X200 Product Family.) 10
- 2.3 Parameter Category Selection 10
- 2.4 Explicit Benchmark Parameter Specification 11
- 2.5 Offload Selection (Only available for the Intel® Xeon Phi™ Coprocessor X200 Product Family.) 11
- 2.6 Coprocessor Selection (Only available for the Intel® Xeon Phi™ Coprocessor X200 Product Family.) 12
- 2.7 Verbosity Selection 12

3 Expert User: Generating Collateral 14
- 3.1 Creating a *micprun_stats* File 14
- 3.2 Data Inspection with micpinfo 14
- 3.3 Data Inspection with micpprint 14
- 3.4 Data Inspection with micpcsv 15
  - 3.4.1 Summary Format 15
  - 3.4.2 Long Format 15
  - 3.4.3 Short Format 16
- 3.5 Data Inspection with micpplot 16

4 Hardware Tester: Executing Regression Tests 18

5 Systems Engineer: Creating Reference Data 20

6 Tinkerer: Benchmark Modifications 21

7 Test Developer: Integrating New Benchmarks 23

8 Summary 24

Appendix A References 25

Figures

Figure 1 Example Summary Output from micpcsv as Displayed by Microsoft Excel 15
Figure 2 Example Output from micpplot..... 17

Tables

Table 1 Benchmarks..... 9
Table 2 Parameter Categories 11
Table 3 Offload Methods 12
Table 4 Verbosity Output 13
Table 5 micprun error codes..... 18

Revision History

Date	Revision	Description
April	1.15	Added information about the Fio benchmark.
March 2016	1.14	Added information about the Deepbench suite.
September 2016	1.13	Added information on the micperf source RPM and how to rebuild it, Sections 2 and 6.
June 2016	1.12	Added new error code in Table 5 Added note to Section 2.2 Memory Selection
June 2016	1.11	Addressed internal feedback
May 2016	1.1	Added HPCG documentation
December 2015	1.0	Addressed internal feedback Added HPLinpack documentation.
November 2015	0.51	Adding 2.2 Memory Selection
June 2015	0.5	Initial release.

§

1 Introduction

There are many benchmarks which can be used with the Intel® Xeon Phi™ Processor X200 Product Family (hereafter referred to as the processor) and the Intel® Xeon Phi™ Coprocessor X200 Product Family (hereafter referred to as the coprocessor) to measure performance. These have different developers, and consequently different user interfaces, different methods of execution and different output. *Micperf* is designed to incorporate a variety of benchmarks into a simple user experience with a single interface for execution and a unified means of data inspection. The user interface to *micperf* consists of five executables: one for execution of benchmarks *(micprun)*, and four that interpret the output of the first one. These executables are all well documented with standard UNIX style command line interfaces. The results can be displayed as professional quality plots, human readable text or comma separated values that can be easily imported into a variety of other applications. Results of different runs can be easily combined and compared. To support tracking of the performance impact of changes to the system configuration, *micperf* stores detailed hardware and software configuration information along with the performance data. *Micperf* also serves as a harness to integrate a variety of benchmarks into automated testing for performance regressions.

The *micperf* package targets a range of users including engineers interested in performance regression testing while implementing modifications to hardware, firmware, drivers or operating system software. In addition to these highly technical customers, there are also application users and hardware manufacturers who use the *micperf* software for demonstration and system verification purposes. This user guide provides a description of the *micperf* user experience for a range of use cases. The simplest use cases are identified in Chapter 2. The sections that follow identify more sophisticated use cases for collateral generation, regression testing, benchmark modification and extending the *micperf* package to include new benchmarks.

The *micprun* executable, the primary application in the *micperf* package, executes several benchmarks: *MKL [12] SMP Linpack [2], MKL HPLinpack [2], MKL HPCG*[1]*, MKL SGEMM, MKL [12] DGEMM, SHOC download*[2]*, SHOC readback*[2]*, STREAM [10][11], Deepbench [16] convolutions*[1] *(libxsmm_layer, std_conv_bench)* and *fio*[1] *[17].* These benchmarks were carefully chosen to demonstrate performance in all of the major bottlenecks in the system. The Vector and Floating Point Unit (VFU) in the processor and the coprocessor excel at dense level three basic linear algebra [9] calculations, and the *Linpack, HPLinpack, HPCG, DGEMM*, and *SGEMM* benchmarks demonstrate these capabilities, *Deepbench* benchmarks exercise operations that are used in deep neural networks. The *fio* benchmark assess performance of I/O operations.

SMP Linpack, HPLinpack, HPCG[1] and *DGEMM* compute with double precision floating point numbers and *SGEMM* computes in single precision. The *SHOC* download[2] and *SHOC* readback[2] benchmarks test the performance of the PCIe bus in transferring data between the host and the coprocessor. The *STREAM* benchmark measures the bus bandwidth between the processor's or the coprocessor's main memory and the computational registers. Deepbench[1] tests performance of convolution operation

[1] Only available for processor.

[2] Only available for coprocessor.

performed using the *Intel® MKL-DNN* and *libxsmm* libraries. Fio[1] checks the performance of drive read operations.

1.1 Organization

This manual is organized into chapters that describe a different user's experience with the *micperf* package.

Chapter 1: Introduction

Chapter 2: Focuses on the final consumer of the processor and the coprocessor products who use basic features of *micperf*.

Chapter 3: Discusses the experience of the advanced user who generates professional quality content about the performance of the processor and coprocessor for presentation.

Chapter 4: Describes the use case of the hardware validator performing performance regression tests against Intel® Validation measurements.

Chapter 5: Details how a systems engineer uses micperf to run performance regression tests on their system modifications.

Chapter 6: Describes how a curious user can inspect or make modifications to benchmark source code.

Chapter 7: Describes a benchmark developer's experience integrating new benchmarks into the *micperf* infrastructure.

1.2 Conventions and Symbols

`This type style`	Indicates an element of syntax, reserved word, keyword, filename, computer output, command, or part of a program example. The text appears in lowercase unless uppercase is significant.
This type style	Used to highlight the elements of a graphical user interface such as buttons and menu names.
This type style	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder. Also used to indicate new terms, URLs, email addresses, filenames, and file extensions.
[*items*]	Indicates that the items enclosed in brackets are optional.
{ *item* | *item* }	Indicates to select only one of the items listed between braces. A vertical bar (|) separates the items.
... (ellipses)	Indicates that you can repeat the preceding item.

§

2 Novice User: Benchmark Execution

An important use case and category, which includes many *micperf* users, is the customer who has just installed a processor or a coprocessor, or has just obtained access to a system that has a processor or coprocessor installed. This customer is interested in the product because of the performance boost it can provide, and they would like to demonstrate the capabilities of this new piece of hardware. *Micperf* provides a demonstration of functionality and performance numbers that serve as a motivation to learn how to use the product.

For the coprocessor, the *micperf* package is distributed as an RPM [3] file within the Intel® Manycore Platform Software Stack (Intel® MPSS). The *micperf* package is distributed as a binary RPM, source code is distributed in the corresponding source RPM. The Intel® MPSS general readme file includes instructions regarding *micperf's* software dependencies and installation instructions.

In the same way for the processor the *micperf* binaries and sources are distributed as a binary RPM and a source RPM respectively within the Intel® Xeon Phi™ Processor Software.

Once installed there is a readme file specific to *micperf* installed to:

/usr/share/doc/micperf-MPSSVERSION/README.txt on RHEL

or

/usr/share/doc/packages/micperf/README.txt on SUSE

This file describes what was installed, basic use instructions and references to other sources of documentation. In the same directory, a change log (`CHANGES.txt`) is posted, and installation instructions are given in a file called `INSTALL.txt`. The *micperf* install procedure conforms to Linux conventions. For the purpose of transparency the source code is distributed in the corresponding source RPM, source code can be inspected, modified and re-build, for more details please refer to Chapter 6.

The first step to using the *micperf* package is running *micprun* and reading the help documentation that the application provides. This is accessed using the POSIX standard [7] getopt style command line arguments in either long form: *micprun --help*, or short form: *micprun -h*. The Unix convention for printing the version number is also respected: *micprun --version*. The command line arguments that give basic control over execution are *-k*, *-c* and *-p*. The amount of output created can be controlled by the verbosity option, *-v*. For more fine-grained control of the output see Chapter 3.

2.1 Benchmark Selection

One of the simplest modifications to the standard micprun execution is to select the benchmarks that will be run (The list of available benchmarks is captured in Table 1.) Benchmark selection is done with the *-k* command line option and by default when *-k* is not specified, all benchmarks are executed. To list the names of all available benchmarks, execute the command *micprun -k help*. More than one benchmark can be selected by a colon-separated list, for example *micprun -k* linpack:dgemm:stream, and the default option to run all workloads can be explicitly specified by *micprun -k* all.

Table 1 Benchmarks

Benchmark	CLI Name	Target Operations	Component
MKL DGEMM[3]	dgemm	Double precision floating point	VFU
MKL SGEMM[3]	sgemm	Single precision floating point	VFU
MKL SMP Linpack	linpack	Double Precision Floating Point	VFU
SHOC Download[1]	shoc download	Bus transfer host to device	PCIe bus
SHOC Readback[1]	shoc readback	Bus transfer device to host	PCIe bus
STREAM[3]	stream	Round-trip memory to registers	MCDRAM, GDDR and caches
HPLinpack[2]	hplinpack	Double precision floating point	VFU
HPCG[2,4]	hpcg	Double precision floating point	VFU
Deepbench convolutions[2]	libxsmm_layer std_conv_bench	Single precision floating point	VFU
Fio[2,5]	Fio	Drive to CPU data transfer	SATA bus

Note: [1] Only available for the coprocessor.
[2] Only available for processor.
[3] For the Intel® Xeon Phi™ processor micperf provides an MCDRAM and a DDR version of this workload. See Section 2.2 for further details on memory selection.
[4] Requires Intel® MPI Libraries, please refer to the INSTALL.txt for details.
[5] The benchmark is not provided with the micperf package, please refer to the *INSTALL.txt* file for details.

2.2 Memory Selection (Only available for the Intel® Xeon Phi™ Processor X200 Product Family.)

MCDRAM memory is one of the key features in the processor, depending on the application the use of MCDRAM memory can help improve performance. The memkind library provides an interface to allocate MCDRAM memory, the SGEMM, DGEMM and STREAM workloads have been modified to demonstrate how performance can be boosted by the use of MCDRAM memory. By default *micprun* will attempt to execute the workloads that allocate MCDRAM memory on failure (e.g. memory mode set to Cache) *micperf* will execute the DDR memory only workloads. -D forces micprun to use directly DDR memory without performing any MCDRAM memory availability check. Please note this option only works in Flat mode as in cache mode MCDRAM is used transparently.

When this option is combined with the –o option (See section 3.1), the name of *micprun_stats* file created by micprun will include the word 'mcdram' or 'ddr' to indicate the type of memory used to execute the workloads.

To learn more about the memkind library please refer to its *manpages*, on Linux after installing the *xppsl-memkind-devel* run on a terminal:

$ man memkind

2.3 Parameter Category Selection

Another basic modification to the standard *micprun* execution is to select the category of parameters that will be passed to the benchmark executables by using the *-c* option. The category names are purposefully abstract, as they are intended to apply to all benchmarks included in *the micperf* package and also to any benchmarks that are added as extensions to *micperf*. The three categories that all benchmarks, including extensions, implement in some form are *optimal*, *scaling*, and *test*. Each benchmark interprets the meaning of these categories differently, but typically *optimal* sets the options that give nearly optimal performance, *scaling* runs at least one parameter through a range of values, and *test* performs a self-check test where *micprun* gives a non-zero return code if the test fails.

For some benchmarks, the exact parameters to achieve optimal performance depend opaquely on the hardware or software configuration. For these benchmarks, several parameter sets may be executed, but if a category begins with the string *optimal* then only the best performing parameters in the set executed are included in *micperf's* summary reporting. This feature allows a benchmark to define a parameter space to search for optimal performance.

The *scaling* category is especially useful for plotting performance as a function of an input parameter to the benchmark. There are some benchmarks that can be scaled in more than one parameter, and for these there are multiple scaling categories. For example, the *DGEMM* benchmark's *scaling* category runs through matrix sizes to show data scaling while using all available cores, but strong core scaling [8] is also implemented as the *scaling_core* category which runs through a range of core counts while keeping the matrix size constant. If a category is selected along with a benchmark that does not implement the category then an error is raised. All of the benchmarks included in the *micperf* package implement the categories defined in Table 2.

Table 2 Parameter Categories

Category	Definition
optimal	parameters that yield nearly optimal performance
scaling	run one parameter, typically data size, through a range of values
test	execute a self-check
optimal_quick	parameters that yield good performance, but with a short run time
scaling_quick	a subset of the scaling category: excludes long run time parameters

2.4 Explicit Benchmark Parameter Specification

The parameter categories serve two functions: one is to abstract the specifics of the benchmark parameters from the user, and the other is to define a common execution purpose for all benchmarks so that they can be run together. There are times, however, when the user wants to have fine-grained control over the parameters that are passed to a particular benchmark. This functionality is accessed with the -p option to *micprun*, and note that the *-p* option cannot be used on the command line with the *-c* option. The benchmarks applications themselves have a number of different ways of obtaining parameters from the user: long form command line options, short form command line options, positionally defined command line arguments, environment variables, or a parameter file. To distill these into a uniform interface, *micprun* takes only long form command line options and maps this specification to the method that each particular benchmark application uses for parameter input.

To print the long form command line options for a benchmark and the default values run: *micprun -k* <benchName> *-p* help.

Note: The default values are the parameters for the optimal category for all of the benchmarks included in the *micperf* package. In the case where the optimal parameter category searches a list of parameters, the default values correspond to the last parameter set in the search list. It is possible to substitute short form command line options by using the first character of the long form option, but this feature can only be used when each of the long form parameters identified with the benchmark starts with a unique character.

2.5 Offload Selection (Only available for the Intel® Xeon Phi™ Coprocessor X200 Product Family.)

Benchmarking the coprocessor differs from benchmarking on other platforms due to the fact that it runs an independent Linux OS communicating with the host system across the PCIe bus. As the name "coprocessor" implies, the use model is to offload work from the host system. There are several methods for doing this and one of the important features of *micperf* is the ability to compare performance characteristics of a variety of offload methods on the same benchmark.

These offload options include running natively on the Intel® Xeon Phi™ coprocessor without any interaction with the host, using the Intel® Symmetric Communications Interface (SCIF) low level API to connect a host and Intel® Xeon Phi™ coprocessor application, and compiler assisted offload where snippets of code from within a host application are offloaded to the Intel® Xeon Phi™ coprocessor during run-time in an automated way. These options can be selected with the -x option to *micprun*, and more than one method can be given in a colon separated list.

Table 3 Offload Methods

Method	Host Process	Device Process	Communication
native	No	Yes	None
scif	Yes	Yes	Intel® Symmetric Communications Interface
pragma	Yes	Yes	Intel® Composer XE Compiler Assisted Offload
auto	Yes	No	None
coi*	Yes	Yes	Intel® Coprocessor Offload Interface
myo*	Yes	Yes	Intel® Shared Memory Library for MPSS stack

Note: * *micperf* does NOT provide any benchmark that uses this offload method but it includes support for custom benchmarks that might use it, see Chapter 6 for details.

2.6 Coprocessor Selection (Only available for the Intel® Xeon Phi™ Coprocessor X200 Product Family.)

Many systems have more than one coprocessor installed. A single instance of *micprun* is able to execute benchmarks on only one coprocessor at a time. The coprocessor index can be selected with the *-d* option. The details about each of the coprocessors and how they map to the coprocessor index can be seen with the Intel® MPSS tool *micinfo*.

2.7 Verbosity Selection

The verbosity option to *micprun* gives coarse control of the output. The *-v* option takes an integer from 0 (least verbose, default value) to 3 (most verbose). The details about the output associated with each level can be found in the *micprun --help* documentation and are outlined in Table 4 Verbosity Output.

In Table 4 the **Summary** column refers to a concluding section in the standard output that reprints the performance data collected as it is displayed by micpprint. The **Joint Plot** column refers to an attempt to create an additional plot that combines the data

from all of the benchmarks. This aggregated plot will only be created in the case where all the selected benchmarks are plotted with the same X and Y axes. The infrastructure supports benchmarks that collect ancillary data that is not displayed by default, but can be displayed if the user requests. The primary data is considered *rolled up*, and the ancillary data is included when all data is requested.

Table 4 Verbosity Output

CLI Options	Summary	Plots	Joint Plot	CSV Rolled	CSV All
-v0	No	No	No	No	No
-v0 -o	No	No	No	No	No
-v1	Yes	No	No	No	No
-v1 -o	Yes	No	No	Yes	No
-v2	Yes	Yes	No	No	No
-v2 -o	Yes	Yes	No	Yes	Yes
-v3	Yes	Yes	Yes	No	No
-v3 -o	Yes	Yes	Yes	Yes	Yes

None of the benchmarks included in the *micperf* package create ancillary data, and consequently *CSV Rolled* and *CSV All* refer to CSV files that have the same content for all included benchmarks.

3 Expert User: Generating Collateral

Some data inspection beyond the benchmark log is available by setting the verbosity level (the *-v* option) of *micprun* above zero. For advanced usage it is recommended that the *-v* option is avoided. More fine-grained control over the output is obtained by using the file created with the *-o* option to *micprun* in conjunction with the *micperf* helper applications: *micpprint, micpplot, micpcsv*, and *micpinfo*.

3.1 Creating a *micprun_stats* File

Understanding how to use the *-o* and *-t* options to *micprun* is the first step to data inspection with the helper applications. The -o option specifies an output directory where output files from *micprun* are created. Setting the verbosity to zero (the default when *-m* is not given) results in only one file being created in the output directory. It will be named *micp_run_stats_TAG.pkl* where TAG is the tag associated with the run. This tag has a default value that describes some of the characteristics of the system under test, but it can be set explicitly with the *-t* option. The tag will be displayed in a number of places in the output, and should be chosen to be descriptive, as well as different from any tags given to runs that are to be compared.

3.2 Data Inspection with micpinfo

A wealth of system information is included in a *micprun_stats* file, and *micpinfo* is the helper application that is used for inspection of this data. The bundling of system information with the performance data allows for the correlation of performance with system configuration and provides a mechanism for comparison of the configurations of two systems that were benchmarked. The system configuration data is collected by running a set of system commands and the standard output of these commands is recorded. The *micpinfo* application allows the user to inspect the log of any subset of the commands using the *--app* option. By default, the application runs the command set on the current system, and if a *micprun_stats* file is passed on the command line then the log produced reflects what was recorded on the system just prior to the execution of the benchmarks when the file was created.

3.3 Data Inspection with micpprint

The benchmark data recorded in a *micprun_stats* file can be displayed in human readable form with the *micpprint* helper application. The output is organized by the benchmark and offload method. Multiple *micprun_stats* files can be passed to the command line of *micpprint*, and the output will be interleaved allowing for easy comparison of the performance recorded in different *micprun_stats* files. Each section of output is preceded by the tag that is associated with the file.

Note: The first file listed on the *micpprint* command line determines the set of benchmark and offload methods that will be displayed. For this reason, when comparing a targeted test to a comprehensive reference file, the targeted test file should be listed

first. This is true for the other helper applications that display performance data as well: *micpcsv* and *micpplot*.

3.4 Data Inspection with micpcsv

The *micpcsv* helper application is used to create data that can be easily parsed by applications external to *micperf*. In particular it produces comma separated value data which is readily imported into a wide variety of applications such as spreadsheets and databases. Since comma separated value output is an unstructured format, there are several styles of output formatting from *micpcsv* which are chosen by passing flags to the command line.

3.4.1 Summary Format

Running *micpcsv* with no arguments produces a single summary table. The data in this table is derived by selecting the highest performance runs for each benchmark, offload method and Intel® Xeon Phi™ coprocessor SKU from the scaling reference data included in the *micperf* package. These data are displayed in a summary table that includes performance information and the value of the independent variable in the scaling parameter category. If the *-o* flag is given then the output file named `summary.csv` is created in the output directory. See Figure 1 for an example of the summary table.

Figure 1 Example Summary Output from micpcsv as displayed by Microsoft Excel

	SGEMM pragma/native (GF/s)	DGEMM pragma/native (GF/s)	SMP Linpack (GF/s)	PCIeDownload pragma/scif (GB/s)	PCIeReadback pragma/scif (GB/s)	STREAM (Triad) (GB/s)
B1QS-5110P	1448.24 / 1668.51	652.43 / 794.54	721.03	6.93 / 6.94	6.98 / 6.98	171.41
Parameters	15360 / 15360	10240 / 7680	26624	1024MB / 1024MB	1024MB / 128MB	58
B1QS-7110P	1550.54 / 1782.89	693.53 / 842.13	755.17	6.99 / 6.95	7.06 / 7.06	175.22
Parameters	15360 / 15360	10240 / 7680	26624	1024MB / 1024MB	1024MB / 1024MB	60

3.4.2 Long Format

If a *micprun_stats* file is passed to *micpcsv* but the *-o* flag is not given then the standard output is composed of blocks of tables which are separated by one or two empty lines. Two tables separated by just one empty line are related: the first is the identifying table, and the second is the run record table. The identifying table has just a header row of the form `KERNEL, OFFLOAD, TAG` and a single row that describes the benchmark, offload method and tag. These three identifiers apply to all of the run records in the table that follows the identifying table. The run record table has a first column of run descriptions. This is followed by columns giving the values of all parameters passed (one column per parameter). The last column in these tables provides the performance metric.

Note: For the optimal parameter category, the summary table for a given kernel will only have two rows: one for the header and one for the values. When specifying the optimal parameter category to *micprun*, the short format output (see Section 3.4.3

Short Format) can be more useful. When specifying the scaling run parameter category to *micprun*, the tables will have multiple rows of values: one for each execution of the benchmark.

If a *micprun_stats* file is passed to *micpcsv* and the *-o* flag is passed then each of the paired tables described above are written to a separate file. The name of each file is derived from the identifying table, and the entirety of the contents of each file is the run record table. This is much more useful than the dump to standard output since each file has a fixed column width.

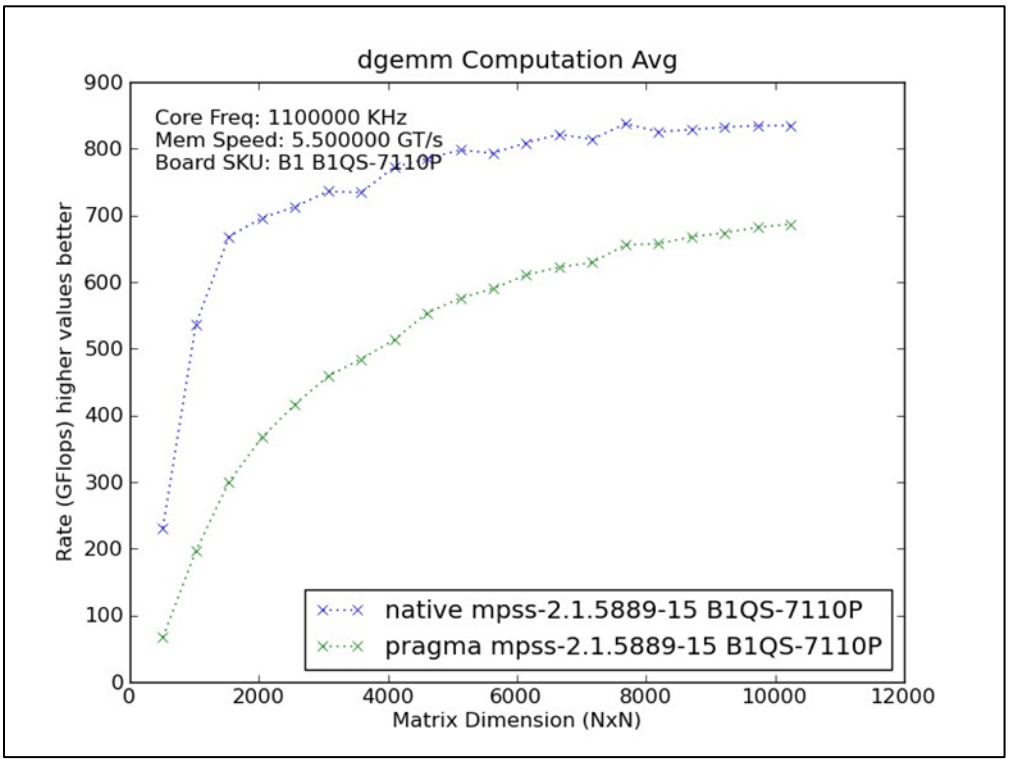
3.4.3 Short Format

The short format is selected by passing the *-s* flag to *micpcsv*. This flag has no impact on the summary format (for instance, when no *micprun_stats* files are given). The short format is advantageous over the long form in that a single table with a fixed column width is produced and disadvantageous in that all of the parameters are stored in a single column and the tags are not included. If the *-o* flag is not given then the table is written to standard output, and if it is given then a single file named `short_form.csv` is produced in the output directory.

3.5 Data Inspection with micpplot

The *micperf* package makes use of the *matplotlib* [6] Python visualization library if it is installed. The *matplotlib* package is not required to be installed on the system being benchmarked, and the *micprun_stats* file can be transferred to another machine for visualization purposes. The *micpplot* application is used to visualize the data from runs using scaling parameter categories, and especially for comparison of different *micprun_stats* files. The application will plot the results from each benchmark on a different graph, and will combine different methods of offload of the same benchmark onto a single graph. As with the other helper applications that take multiple *micprun_stats* files, the first file listed determines the benchmarks and offload methods that will be plotted. In the case where all of the benchmarks that are plotted have the same x and y axis then *micpplot* will produce a final image that plots all of the benchmarks onto a single graph. By default the *micpplot* application generates interactive plots that the user can re-size and save in the Portable Network Graphics (PNG) [5] format with a user specified file name. As the user closes each plotting window, the next one in the sequence appears. The *micpplot* application also accepts the *-o* option which creates PNG files in the specified directory in a non-interactive mode. An example of the output from *micpplot* is given in Figure 2.

Figure 2 Example Output from micpplot



4 Hardware Tester: Executing Regression Tests

The *micperf* package is distributed with a set of reference *micprun_stats* files. These files are accessible by using the -R command line option with *micprun* and all of the *micperf* helper applications. When an application is called with *-R help* it will print the list of tags that are available in the installed location. The user can specify one of the tags from the list as the argument to the -R option. This feature is especially useful for performing regression tests with *micprun*. The data stored in the distributed reference files was measured by Intel® Validation and can be used as a reference mark to determine if the Intel® Xeon Phi™ processor or Intel® Xeon Phi™ coprocessor are performing up to specification. To execute this test regression the *-m* option can be used to specify the acceptable margin allowed from the reference measurement given as a percentage. If the performance measured by this run is lower than the reference by more than the fractional margin an error message is printed and the return code from *micprun* is 88, return codes are documented in the *micprun --help* output, for convenience the rest of the error codes are also presented in Table 5.

Table 5 micprun error codes

Error Code	Meaning
0	No error
1	Unhandled python exception
2	Command line parse error
3	File I/O error
88	Performance regression error
89	MPSS service not available error
90	Kernel or offload lookup error
91	Linpack kernel could not be executed (missing dependencies).
127	Missing shared object libraries error

The data that is distributed uses the default tag which contains the product stock keeping unit (SKU), the Intel® MPSS version (for the coprocessor) or the Intel® Xeon Phi™ processor software version (for the processor), the offload methods and the parameter category. The Unix grep utility can be used to select the tag appropriate for the regression test to be performed. For instance, to run a regression test on a 5110 SKU part using the optimal parameter category the user would run the following command:

```
$ refTag=‘micprun -R help | grep 5110 | grep optimal‘ micprun -t test -o
. -m 0.04 -R $refTag
```

Note: The *-o* and *-t* flags were given above which produces a *micprun_stats* file. It can be very useful to inspect this output if a performance regression is detected.

Note: It is possible to run a subset of the benchmarks included in the reference file by specifying the *-k* and/or *-x* flags (in fact all flags can be overridden).

To show how the *micprun_stats* file from the regression test can be used with the helper applications we will give some examples here. The *micpinfo* application can be used to compare the difference between the two system configurations using the Unix diff tool:

```
$ micpinfo -R $refTag > ref_info.log
$ micpinfo micp_run_stats_test.pkl > test_info.log
$ diff ref_info.log test_info.log > diff_info.log
```

This will highlight any differences between the *micpinfo* logs and put this comparison into the file *diff_info.log*. This comparison can be narrowed down by passing the *--app* flag to *micpinfo*: for example*,* passing *--app conf* in the calls to *micpinfo* above will show the differences in the mic configuration files. The *micpprint* utility can be used to examine the performance data and compare with the reference:

```
$ micpprint -R $refTag micp_run_stats_test.pkl
```

Similar results can be produced in a form more easily parsed by a computer program using *micpcsv*. Note that the tags are not displayed in the CSV short form output making the *-s* flag less useful when comparing multiple files. The *micpplot* application can be used to inspect the regression; for example, the command:

```
$ micpplot -R $refTag micp_run_stats_test.pkl
```

will plot the test lines against the reference lines.

Note: The identifying tag is displayed in the plot legend.

5 Systems Engineer: Creating Reference Data

Chapter 4 showed how to use the reference data shipped with *micperf* to perform regression tests. The way the user accesses the reference data in this case is through selecting a tag with the *-R* flag, but all of the examples in Chapter 4 can be executed with user created reference files as well. User-created reference files can be used to determine how performance changes as an element of the system under test, as opposed to complete system comparison against Intel validation measurements. A simple example of this would be BIOS settings, and this example can be easily extended to apply to changes to the operating system software.

Let's say the user wants to determine if changing a power management BIOS setting on the host system will have an impact on performance. To test this, the user generates a *micprun* stats file on the system with the original BIOS setting using the *-o* option to *micprun*. The user then reboots the system, changes the BIOS setting and then runs against the *micprun* stats file just created using the *-r* and *-m* options to *micprun*. In this way, the user can isolate how specific changes in a system impact performance. The BIOS setting is just one example of a change to a platform. This change could be a software modification to the kernel or driver, or any change in the design of a platform that includes a processor or a coprocessor. Any engineer involved in the design of the platform that wants to elucidate the performance impact of a design change can use the tool in this way.

6 Tinkerer: Benchmark Modifications

Transparency is an important consideration in the distribution of benchmarking software and one of the most important aspects of this transparency is using open source benchmarks that are standards in the industry. With the exception of the Intel® MKL benchmarks (SMP Linpack, HPLinpack and HPCG), all of the benchmarks that are included in the *micperf* package are distributed with source code and make files for compilation. The source files are included in the corresponding source RPM: xppsl-micperf-<version>-<release>.src.rpm or mpss-micperf-<version>-<release>.src.rpm depending on the software stack, source RPMs can be re-build to generate a new binary RPM. Source code can be inspected and/or even modified an example of how to recompile the micperf source RPM will be presented below, for further details on how to apply a patch and re-build the soruce RPM please refer to [14] and [15]. Intel® MKL benchmarks can be used with *micperf*; however, the binary is distributed by Intel® MKL, and neither the source nor the binary are distributed with the *micperf* package.

To inspect the source code "install" the source RPM and change to the appropriate directory (please note commands are executed as a non-root user):

```
$ rpm --ihv xppsl-micperf-<version>-<release>.src.rpm

$ cd ~/rpmbuild/SOURCES/

$ tar -xf xppsl-micperf-<version>.tar.gz

$ cd xppsl-micperf-<version>
```

In order to re-build the source RPM, the Intel® Composer XE package must be installed, and the *compilervars* script must be sourced using the following command for bash:

```
source /PATH/TO/COMPOSER_XE_INSTALL_DIR/bin/compilervars.sh intel64
```

or for C shell:

```
source /PATH/TO/COMPOSER_XE_INSTALL_DIR/bin/compilervars.csh intel64
```

If these requirements are met then the user can simply run:

```
$ rpmbuild --rebuild mpss-micperf-<version>-<release>.src.rpm   # coprocessor
```

Or

```
$ rpmbuild --rebuild xppsl-micperf-<version>-<release>.src.rpm # processor
```

Please note the commands above are executed as non-root user, the binary RPM produced by *rpmbuild* is typically stored in the *<HOME*

DIRECTORY>/rpmbuild/RPMS/x86_64 directory, the exact location and name of the binary RPM are reported by rpmbuild in its output, look for the line:

Note: *"Wrote: /PATH/TO/BINARY/RPM.rpm"* When using *rpmbuild* and the spec file to re-build the source RPM, it is needed to define three variables in the command line as follows:

```
$ rpmbuild --bb <SPEC file> --define 'name <NAME>'  --define 'version
<VERSION>' --define 'release <RELEASE>'
```

Where:

- <NAME> is xppsl-mipcerf or mpss-micperf depending on the software stack
- <VERSION> for instance 1.5.0 or 4.4.3
- <RELEASE> a single digit indicating the release number e.g. 1

For a concrete example:

```
$ rpmbuild --bb xppsl-micperf-1.5.0.spec --define 'name xppsl-micperf'  -
-define 'version 1.5.0' --define 'release 1'
```

Note: The *micperf* make files respect the GNU standard *DESTDIR* and *prefix* variables that can be used to relocate the install path; however, if the install path is relocated *micprun* will not find the new executable at run time.

An example of a user modification to benchmark source code is changing the GEMM benchmark so that it uses *malloc* rather than *mmap* or *memkind_posix_memalign* to allocate the memory used for the computation. The user could replace the *mmap* call in the *utils.c* file in the GEMM source directory, generate a patch and rebuild the source RPM to produce a new binary RPM. This could even be done as a regression test following the steps outlined in Chapter 5, where the benchmark is modified rather than an element of the platform.

7 Test Developer: Integrating New Benchmarks

The *micperf* package is designed to be able to incorporate a wide range of benchmarks, and can be used to wrap small computational kernels that are not fully featured benchmarks. The *micperf* infrastructure is written in Python and the class that is used for abstracting the requirements of a benchmark or computational kernel is called *micp.kernel.Kernel*. If a user wants to extend the set of benchmarks used by *micprun* they simply need to add a package to the Python environment (typically with the *PYTHONPATH* environment variable, or by installing into the *site-packages*) where this package has one module for each add-on benchmark, and each module has a class that inherits from *micp.kernel.Kernel*. Each user defined class derived from the *micp.kernel.Kernel* class must have the same name as the module that includes it. To access a user defined add on package with *micprun*, the package name is passed with the *-e* option. The *kernels* sub-directory of the *micp* package serves as an example of how the add-on package should be structured including the *__init__.py* package file.

The *micp.kernel.Kernel* class is an abstract base class with a supporting factory class [4]. Each method of the base class has a doc string that defines the requirements of the method, and each method has a default implementation with the exception of *__init__()*. The default implementations are designed to support simple computational kernel function calls that were wrapped with an executable that uses positional command line arguments and prints performance data with a format styled after the Google Test framework. To the extent that a workload deviates from this simple case the base class methods must be overridden.

The default implementations provided by the Kernel base class should not be used when adapting an existing benchmark if doing so would require alteration of the benchmark; rather, the method implementations of the derived Kernel class should be adapted to the behavior of the existing benchmark. The requirements and definitions of the class methods can be obtained by looking at the help for this class. This can be done by invoking a Python interpreter that has *micp* included in the PYTHONPATH and run:

```
>>> import micp.kernel

>>> help(micp.kernel.Kernel)
```

The user can also refer to the kernels sub-directory of *micp* which contains examples of how the class was adapted to run the benchmarks included in *micp*.

8 Summary

The *micperf* package provides benchmarking solutions for the users and developers of the processor and coprocessor. The material presented here gives a holistic view of the users and use cases, but is not a substitute for the specific detailed documentation provided by the --help message from the *micperf* executables or the other documentation included in the package. In this paper the reader has learned how to run benchmarks and create presentation material from those runs. The paper has covered how to do regression testing against Intel validation performance measurements, and how to do regression testing against the user's own measurements. Some of the details on how to modify the source code for the included benchmarks and how to extend *micperf* to include new benchmarks were discussed.

The *micperf* package appeals to community standards for benchmark inclusion, open source distribution, GNU standard build, Unix command line interfaces and established object oriented design patterns. The use of standards allows the *micperf* package to be intuitive for users who are familiar with the Unix environment and industry benchmarks. Appealing to standards also improves transparency while lending credibility to the results. The *micperf* package allows for modular extension by using the object oriented design offered by Python while decoupling the Python infrastructure from the build of the benchmark executables. The users and use cases discussed cover a wide range of benchmarking needs, and this paper will facilitate wider and more effective use of the tool.

§

Appendix A References

[1] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pages 63–74, New York, NY, USA, 2010. ACM.

[2] Jack Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[3] E. Foster-Johnson. *Red Hat RPM Guide*. Linux solutions from the experts at Red Hat. Wiley, 2003.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[5] Michael J. Hammel. Png: The definitive guide. *Linux J.,* 2000(69es), January 2000.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering,* 9(3):90–95, 2007.

[7] *IEEE. IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Shell and Utilities, Issue 6*. 2001. Revision of IEEE Std 1003.11996 and IEEE Std 1003.2-1992) Open Group Technical Standard Base Specifications, Issue 6.

[8] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[9] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.,* 5(3):308–323, September 1979.

[10] John D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http://www.cs.virginia.edu/stream/.

[11] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter,* pages 19–25, December 1995.

[12] Intel® Math Kernel Library (Intel® MKL). http://software.intel.com/en-us/intel-mkl.

[13] https://software.intel.com/en-us/articles/intel-optimized-technology-preview-for-high-performance-conjugate-gradient-benchmark

[14] https://wiki.centos.org/HowTos/RebuildSRPM

[15] https://fedoraproject.org/wiki/Packaging:Guidelines?rd=Packaging/Guidelines#Patch_Guidelines

[16] https://github.com/baidu-research/DeepBench

[17] https://github.com/axboe/fio

§